多最小效用阈值的频繁高效用项集快速挖掘算法 *

王　斌，吕瑞瑞，房新秀，马俊杰

(青岛理工大学 信息与控制工程学院，山东 青岛 266033)

摘　要：针对多最小效用阈值高效用项集挖掘算法(MHUI)中出现的重复计算、挖掘的结果项集不是频繁的问题，提出两个新的快速挖掘算法 FMHUI 和 SFMHUI。FMHUI 算法在计算项集的最小效用阈值时利用前一次计算结果，避免了项之间的重复比较。另外定义了项的扩展项的最小效用阈值表 EMMU-table 快速计算出扩展项的最小效用阈值，提高了运行效率。SFMHUI 算法在 FMHUI 的基础上增加了支持度约束，使挖掘的项集既是高效用的也是频繁的。通过仿真实验验证了所提出算法的高效性和可行性。

关键词：频繁项集；高效用项集；支持度；多最小效用阈值

中图分类号：TP301.6　　doi: 10.3969/j.issn.1001-3695.2018.06.0403

Fast mining algorithm for frequent and high utility itemsets with multiple minimum utility thresholds

Wang Bin, Lyu Ruirui, Fang Xinxiu, Ma Junjie

(*School of Information & Control Engineering*, *Qingdao Technological University*, *Qingdao Shandong* 266033, *China*)

Abstract: In mining algorithm for high utility itemsets with multiple minimum utility threshold (MHUI) , calculation is often repeated and mining result itemsets are not frequent . This paper developed two new fast mining algorithm SFMHUI and FMHUI. The FMHUI algorithm used the previous calculation result in the calculation of the minimum utility threshold of the itemsets, avoiding duplicate comparisons between items; in addition, the FMHUI algorithm defined the minimum utility threshold table EMMU-table of extensions of items to quickly calculate the minimum utility threshold of extensions, improving the efficiency. The SFMHUI algorithm added the support constraints on the basis of FMHUI, making the mining itemsets both high-utility and frequent. The result from simulation experiments shows that the proposed algorithms are efficient and feasible.

Key words: frequent itemsets; high utility itemsets; support; multiple minimum utility thresholds

0　引言

挖掘高效用项集是 Chan 等人[1~4]在 2003 年首次提出的，近几年来挖掘高效用项集在现实生活中应用非常广泛，如市场购物篮分析[4,5]、网站点击流分析[6]、零售商店中的交叉营销、医疗保健和生物医学应用等[5,7,8]。

最初，高效用项集挖掘算法都是采用单一效用阈值，如 HUC-Prune[9] 、 UP-Growth+[10] 、 MU-Growth[11] 、 HUI-Miner [12],FHM[13]、HUP-Miner[14]、EFIM[15]算法等。但是采用单一效用阈值没有考虑项的多样性，这在现实生活中是不适当的和不公平的。因此采用多最小效用阈值挖掘高效用项集应运而生，它考虑了项之间的差异，对每一个项都赋一个最小效用阈值，解决了单一效用阈值的不足。多最小效用阈值挖掘算法主要有文献[16]中基于 Apriori[17]算法的 HUI-MMU、HUI-MMU$_{TID}$、HUI-MMU$_{TE}$，它们与 Apriori 算法一样产生了过多的候选项集和需要多次扫描数据库；文献[18]基于多最小项效用集合枚举树 MIU-tree(multiple item utility Set-enumeration tree)和效用列表的 HIMU、HIMU-EUCP 算法，避免了额外的数据库扫描和候选项集产生。以上两类算法都有一个共同的特点，项的处理顺序是按照项最小效用阈值递增排序，避免了在求项集的最小效用阈值时项最小阈值之间的比较。后来，Krishnamoorthy[19]提出多最小效用阈值挖掘算法 MHUI (high utility itemsets with multiple minimum utility thresholds)，它采用效用列表结构并通过四种剪枝策略来提高挖掘效率，但是 MHUI 算法的项的处理顺序是按照事务权重效用 TWU[18~20](transaction weighted utility)递增排序的,并且验证了采用 TWU 进行排序处理比采用项的最小效用阈值进行排序效率更高（因为采用项最小效用阈值排序虽然避免了项最小阈值之间的比较，但是会导致候选项集数量

收稿日期：2018-06-25；**修回日期：**2018-08-27　　**基金项目：**国家自然科学基金资助项目（61502262）

作者简介：王斌（1963-），男，山东青岛人，硕导，主要研究方向为知识发现、博弈论及应用（wb769@sina.com）；吕瑞瑞（1991-），女，硕士研究生，主要研究方向为数据挖掘、知识发现；房新秀（1994-），女，硕士研究生，主要研究方向为数据挖掘、知识发现；马俊杰（1992-），男，硕士研究生，主要研究方向为数据挖掘、知识发现.

的增加，而按 TWU 进行排序产生更少的候选项集，从而使效率更高）。然而 MHUI 算法在求每个项集的最小效用阈值和扩展项的最小效用阈值时，项集及其扩展项的最小效用阈值都要重新比较，进行了多次重复计算，降低了挖掘效率。针对 MHUI 算法的这个问题，本文提出 w 多最小效用阈值的高效用项集快速挖掘算法 FMHUI（fast mining high utility itemsets with multiple minimum utility thresholds ），并结合四个剪枝性质加以改进。

以上的这些算法只能保证挖掘的项集是高效用的，对项集是不是频繁的并不能确定。众所周知，高效用项集挖掘和频繁项集挖掘在数据挖掘领域都有着非常重要的意义[17,21]。但是大多数的研究要么使用支持度约束来挖掘频繁项集，要么使用效用约束来挖掘高效用项集。然而单独考虑这两种约束都有各自的局限性，比如支持度很高的项集其效用不一定高，同样效用很高的项集其支持度也不一定高。由表 1 和 2 知(设最小支持度阈值 2/6),项集 af,其效用值 U(af)=18>15 是高效用项集，但是其支持度 sup(af)=1/6<2/6 不是频繁项集，而项集 ad 其支持度 sup(ad)=2/6=2/6 是频繁项集，但是其效用值 U(ad)=24<25 不是高效用项集。针对这个问题，本文提出加上支持度约束，并在改进的 FMHUI 算法的基础上提出有支持度约束的多最小效用阈值高效用项集快速挖掘算法 SFMHUI（FMHUI with support constraints），解决了 MHUI 算法的重复计算和挖掘的结果不是频繁项集的问题。

1 相关定义及问题陈述

给定 $I=\{i_1,i_2,...,i_m\}$ 是一个含有 m 个不同项的集合，任意事务 $T_j=\{x_l \mid l=1,2,...,N_l, x_l \in I\}(1\le N_j \le m)$ 是 I 的一个子集。对 T_j 中任意 $x_i \in T_j$，都由外部效用 $EU(x_i)$ 和内部效用 $IU(x_i, T_j)$表征；数据库 $D=\{T_1,T_2,...,T_n\}$(n 是事务的总数)是所有事务的集合；项集 $X=\{x_1,x_2,...,x_k\}\subseteq I$ 被称为 K-项集。一个样本数据库 D 如表 1 所示，每个项的外部效用 EU(x_i) 、最小效用 MU(x_i)和事务权重效用(TWU)如表 2 所示。

定义 1 项 x_i 在 T_j 中的效用用 $U(x_i,T_j)$表示，定义为

$$U(x_i,T_j)=EU(x_i)*IU(x_i,T_j) \tag{1}$$

定义 2 项集 X 在事务 T_j 中的效用用 U(X,Tj)表示，定义为

$$U(X,T_j)=\sum_{x_i\in X}U(x_i,T_j) \tag{2}$$

定义 3 项集 X 在数据库 D 的效用用 U(X)表示，定义为

$$U(X)=\sum_{X\subseteq T_j\in D}U(X,T_j) \tag{3}$$

定义 4 事务 T_j 的效用用 TU(T_j)表示，即

$$TU(T_j)=\sum_{X\subseteq T_j\wedge x_i\in X}U(x_i,T_j) \tag{4}$$

定义 5 项集 X 的事务权重效用用 TWU(X)表示,定义为

$$TWU(X)=\sum_{X\subseteq T_j\in D}TU(T_j) \tag{5}$$

定义 6 所有项的最小效用阈值用 MMU(multiple minimum utility)表示，定义为

$$MMU=\{MU(x_1),MU(x_2),...,MU(x_m)\} \tag{6}$$

其中：m 为不同项的个数。如数据库 D 的 MMU={25,30,17,28,22,15}

表 1 样本数据库 D

Tid	项 (item)	内部效用 (IU)	效用 (U)	事务效用 (TU)
T1	a, c, d, f	3, 2, 3, 2	12, 6, 3, 6	27
T2	a, d, e	2, 1, 2	8, 1, 18	27
T3	a, b, c, e	1, 2, 2, 1	4, 16, 6, 9	35
T4	b, d, e	1, 3, 2	8, 3, 18	29
T5	b, c, d	3, 5, 2	24, 15, 2	41
T6	b, c, d, e	3, 2, 4, 1	24, 6, 4, 9	43

定义 7 用 LMU(least minimum utility threshold)表示所有项的最小最小效用阈值，定义为

$$LMU=\min\{MU(x_1),MU(x_2),...,MU(x_m)\} \tag{7}$$

定义 8 k-项集 X={$x_1, x_2, . . ., x_k$}的最小效用阈值用 MIU(X_k)表示，定义为

$$MIU(X_k)=\min\{MIU(X_{k-1}),MU(x_k)\} \tag{8}$$

定义 9 项的排序。把所有项按项的事务权重效用进行递增排序，通过表 2 可以看出 $f\prec a\prec e\prec c\prec b\prec d$。数据库 D 项的重新排序见表 3(因 sup(f)=1/6<2/6,所以接下来在使用数据库 D 时 f 已被删除，即表 3)。

定义 10 在 T_j 中项集 X 之后的项，用 T_j/X 表示，其效用称为 X 的剩余效用用 RU(T_j/X)表示，定义为

$$RU(T_j/X)=\sum_{x_i\in(T_j/X)}U(x_i,T_j) \tag{9}$$

例如,RU(T_6/ec)=U(b,T_6)+U(d,T_6)=28

表 2 项的效用信息

item	a	b	c	d	e	f
EU	4	8	3	1	9	3
MU	25	30	17	28	22	15
TWU	89	148	146	166	134	27

定义 11 数据库中的项按定义 9 排序后，定义排在项 x_i 之后的所有项称为项 x_i 的扩展。项 x_i 及 x_i 的扩展所对应的最小效用阈值用 $EMU(x_i)$表示，数据库中所有项的扩展项最小效用阈值 EMU(extended minimum utility threshold of items)用扩展项的多最小效用阈值表 EMMU-table(extended multiple minimum utility threshold table for items)表示,定义为

$$EMMU-table=\{EMU(x_1),EMU(x_2),...,EMU(x_m)\} \tag{10}$$

需要特别指出的是，求 EMMU-table 时是按照排序后的项倒序求每个项的 EMU。例如项的排序是 $a\prec e\prec c\prec b\prec d$，先求 EMU(d)=28，然后 EMU(b)=min{EMU(d),MU(b)}=28；同理，EMU(c)=min{EMU(b),MU(c)}=17,EMU(e)=17,EMU(a)=17,因此表 1 的 EMMU-table={17,17,17,20,20}。

定义 12 利用项集的扩展，引入一个全局最小效用阈值

GMU（global minimum utility threshold），定义为

$$GMU(X)=\min\{MIU(X),EMU(x_k)\} \tag{11}$$

其中：x_k 为项集 X 的最后一项。例如：

GMU(ac)=min{MIU(ac),EMU(c)}=min{17,17}=17

表 3　排序处理后的数据库 D

Tid	项 (item)	效用 (U)	事务效用 (TU)
T1	a, c, d	12, 6, 3	21
T2	a, e, d	8, 18, 1	27
T3	a, e, c, b	4, 9, 6, 16	35
T4	e, b, d	18, 8, 3	29
T5	c, b, d	15, 24, 2	41
T6	e, c, b, d	9, 6, 24, 4	43

定义 13　项集 X 在数据库中出现的事务数占总事务数的比率，称为项集 X 的支持度，用 $sup(X)$表示。

性质 1　如果项集 X 的支持度 sup(X)<minsup，那么 X 的所有超集的支持度都小于 minsup,该性质称为项集的反单调性[17]。

定义 14　数据库中所有项组成的 2-项集支持度矩阵用预估支持度共现结构 ESCS（estimated support co-occurrence structure ）表示，定义为

$$ESCS(X=\{x_i,x_j\})=\sup(X=\{x_i,x_j\}) \tag{12}$$

这是个三角阵用来存储数据库中所有 2-项集的支持度，可参照预估效用共现结构 EUCS(estimated utility co-occurrence structure)[13,14]。

问题陈述：本文目的就是在数据库中挖掘所有频繁高效用项集 FHUIs（frequent and high utility itemsets），即

$$FHUIs=\{X:U(X):\sup(X)\,|\,X\subseteq I,U(X)\geq MIU(X),\ \sup(X)\geq \min\sup\} \tag{13}$$

2　SFMHUI 算法

因为 SFMHUI 算法是 FMHUI 算法加上支持度的改进，所以本章主要介绍 SFMHUI 算法的搜索空间、剪枝性质以及算法的主要流程。在此之前先介绍算法的数据存储结构，可参照文献[12~14,19]所介绍的效用列表结构来存储项集信息,并在此基础上加上支持度相关的统计信息，即每个项集的 Tid 的总数。其初始（即 1-项集）效用列表如图 1 所示。

a(25)

3	24	59
1	12	9
2	8	19
3	4	31

e(22)

4	54	75
2	18	8
3	9	22
4	18	11
6	9	34

c(17)

4	33	73
1	6	3
3	6	16
5	15	26
6	6	28

b(30)

4	72	9
3	16	0
4	8	3
5	24	2
6	24	4

d(28)

5	20	0
1	3	0
2	8	0
4	3	0
5	2	0
6	4	0

图 1　初始效用列表

2.1　搜索空间

高效用项集挖掘问题的搜索空间都可以用一个集合枚举树表示。以数据库 D(表 3)为例，其项 $x_i\in I=\{a,b,c,d,e\}$ 且 $a\prec e\prec c\prec b\prec d$，则 I 的所有项集用集合枚举树表示，如图 2 所示。SFMHUI 算法挖掘时采用深度优先搜索，因此 k+1-项集的最小效用阈值可以用 $\min\{MU(x_{k+1}),MIU(X_k)\}$,而不是每次都重新比较项集中所有项的最小效用阈值 ($\min\{MU(x_1),MU(x_2)...MU(x_{k+1})\}$其中 k=1,2,3....)，从而减少了重复计算，提高了算法的运行效率。

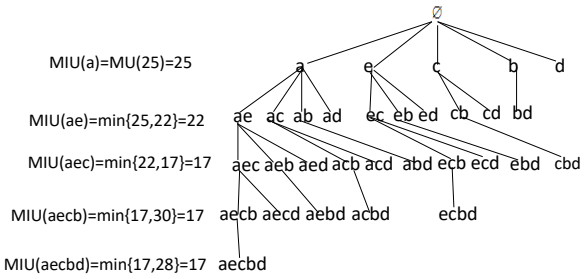


图 2　集合枚举树

2.2　主要的剪枝策略

SFMHUI 算法挖掘的项集是频繁高效用项集，即挖掘出的项集其支持度和效用值都必须满足最小的约束条件。因此可参照高效用项集挖掘和频繁项集挖掘中的一些剪枝性质[13,16~19,21,22]，但是它们有的并不能直接使用，需要转换成适合 SFMHUI 算法的剪枝性质。下面是主要的剪枝性质：

性质 2　Item-Prune。如果有项 x_i 其 $TWU(x_i)<LMU$ 或者 $\sup(x_i)<\min\sup$，那么由项 x_i 组成的所有项集都不可能是频繁高效用项集。

性质 3　SU-Prune。如果一个项集 X，其 $U(X)+RU(X)<GMU(X)$ 或者 $\sup(X)<\min\sup$ 那么有 $\forall X`\supseteq X,X`\notin FHUIs$。

证明　$U(X`)=\sum_{T_j\in D}U(X`,T_j)$

$$\leq\sum_{T_j\in D}U(X)+RU(X)$$

$$\leq U(X)+RU(X)$$
$$<GMU(X)$$
$$<MIU(X)$$

其支持度约束根据性质 1 可证。因此，X 及 X 的超集都不是频繁高效用项集。

性质 4　EC-Prune。如果 $EUCS(X)<GMU(X)$ 或者 $ESCS(X)<\min\sup$，那么没有 X 的超集是频繁高效用项集。

性质 5　给定两个项集 X 和 Y，如果 $U(X)+RU(X)-\sum_{\forall T_j\in D,X`\subseteq T_j`\wedge Y\not\subset T_j}U(X,T_j)+RU(X,T_j)<GMU(X)$,那么 $\forall Y`\supseteq Y\wedge X`\supseteq X,X`Y`\notin FHUIs$

证明参照文献[19]。

在算法运行过程，通过使用上面四种剪枝策略，可以有效地提高频繁高效用项集挖掘的效率。接下来介绍 SFMHUI 算法。

2.3　SFMHUI 算法介绍

SFMHUI 算法重新定义了最小效用阈值的计算方法，即利

用前一次计算的结果，避免前面已经计算的项集的最小效用阈值再重复计算的问题。并且利用求出的 EMMU-table 快速得到项的扩展项的最小效用阈值，提高了求项集的全局效用 GMU 的效率，从而提高了算法的运行效率。加上支持度约束使挖掘的项集是频繁高效用项集。下面介绍详细算法：

算法 1 以事务数据库 D、最小支持度阈值 minsup 和所有项的最小效用阈值 MMU 作为输入。算法总共扫描数据库两次，第一次扫描后计算出所有 1-项集的事务权重效用 TWU 和支持度值 sup；第二次扫描时（算法 5、6 步）做的工作主要有：a）利用 Item-Prune 剪枝没有希望的 1-项集(性质 2 剪掉的项)并重新计算所有事务的事务效用 TU（transaction utility）；b）为所有有希望(即 I^*中的项）的 1-项集构建 EUCS、ESCS 结构以及 1-项集效用列表（图 1）。然后，产生的 1-项集效用列表用于产生项集搜索树并挖掘频繁高效用项集.

算法 1 SFMHUI 算法：main

输入：数据库 D，所有项的最小效用阈值 MMU，最小支持度阈值 minsup。

输出：频繁高效用项集 FHUIs。

1 Scan D and compute TWU and sup for all 1-itemsets

2 $I^* \leftarrow$ each and item x^i such that $TWU(x_i) \geq LMU$ and $\sup(x_i) \geq \min\sup$ /* 性质 2 */

3 Let $\succ$ be the total order of TWU ascending values on I^* and caculate EMMU -table

4 For each $T_j \in D$ sort $x_i^{'}s$ in T_j as per TWU ascending order and recompute TU

5 Iteratively construct 1-itemset($x_i \in I^*$) ULs and EUCS structure and ESCS structure

6 FHUIs=Explore-Search-Tree({},ULs,MMU,minsup)

/*算法 2 */

算法 2 和 3 主要是项集搜索树的挖掘过程和项集效用列表的构建过程。其中，算法 2 主要利用剪枝性质 3、4 进行剪枝搜索空间，算法 3 主要介绍项集列表的构建过程[12,14,18]和剪枝策略 5 的应用。

算法 2 SFMHUI：Explore-Search-Tree

输入： 以项集 P 为前缀的的效用列表 P，所有 P 的 1-扩展效用列表集 ULs(utility-lists)，MMU，minsup，EMMU-table。

输出：以 P 为前缀的所有频繁高效用项集 FHUIs。

1 for each utility list X in ULs do

2 if $U(X) \geq MIU(X)$ and $\sup(X) \geq \min\sup$ then $FHUIs = \{FHUIs \cup X\}$

3 if $U(X)+RU(X) \geq GMU(X)$ and $\sup(X) \geq \min\sup$ then /* 性质 3*/

4 $exULs \leftarrow \{\}$

5 for each utility list Y after X in ULs do

6 if $EUCS(X-P,Y-P) \geq GMU(X)$ and $ESCS(X-P,Y-P) \geq \min\sup$ then

7 /* 性质 4*/

8 $UL(XY) =$ Construct-Utility-List(P,X,Y)

9 / * 算法 3 */

10 if UL(XY) ≠ NULL then

11 $exULs = \{exULs \bigcup UL(XY)\}$

12 end if

13 end if

14 end for

15 Explore-Search-Tree(X,exULs,MMU,minsup, EMMU-table)

16. end if

17. end for

算法 3 SFMHUI: Construct-Utility-List

输入：项集 P，Px，Py 的效用列表。

输出：项集 Pxy 的效用列表。

1 UL(Pxy) = NULL

2 set TUTIL(Px)=U(Px)+RU(Px) /* 性质 5 */

3 for each $ex \in Px$ do

4 if $\exists ey \in Py$ and ex.tid==ey.tid then

5 if P is not empty then

6 Search element $e \in P$ such that e.tid==ex.tid

6 exy=<ex.tid,U(ex,ex.tid)+U(ey,ey.tid)-U(e,e.tid), RU(ey,ey.tid)>

7 else

8 exy=<ex.tid,U(ex,ex.tid)+U(ey, ey.tid), RU(ey,ey.tid)>

9 end if

11 UL(Pxy)={UL(Pxy) ∪exy}

12 创建或者更新 UL(Pxy)

13 else /* 重新计算 TUTIL，并利用性质 5*/

14 TUTIL(Px)=TUTIL(Px)-U(Px,ex.tid)-RU(Px,E.tid)

15 if TUTIL(Px)<GMU(Px) then return NULL

16 end if

17 end for

18 return UL(Pxy)

3 实验分析

为了验证提出算法的高效性和可行性，本文进行两组仿真实验：MHUI 和 FMHUI 算法。SFMHUI 和 SMHUI 算法(SMHUI 算法是在 MHUI 算法的基础上加上与 SFMHUI 算法相同的支持度约束条件以及相关的频繁项集剪枝策略)。这两组对比实验的目的：一是验证在相同的条件下改进后的算法效率更高；二是验证算法加上支持度约束后算法是可行的。实验环境如下：操作系统 Windows7(64 位)，开发工具 eclipse，编程语言为 Java，4 GB 内存容量，英特尔 i5 处理器。

表 4　数据集特征

数据集	事务数量	项数(I)	平均长度(A)	密度(A/I) %
Mushroom	8124	119	23	19.3
Accidents	340183	468	33.8	7.2

3.1　实验设计

通过在 Accidents 和 Mushroom 上进行评估实验。这两个数据集都是从 SPMF[23]文库中下载的。这两个数据集的特征如表 4 所示。

项的内部效用在 1~10 整数之间随机产生，外部效用使用对数正态分布在 0.01~10 之间随机产生，可参照文献[12~14]。每个项赋最小阈值，参照文献[16,18,19]中的赋值方式，即

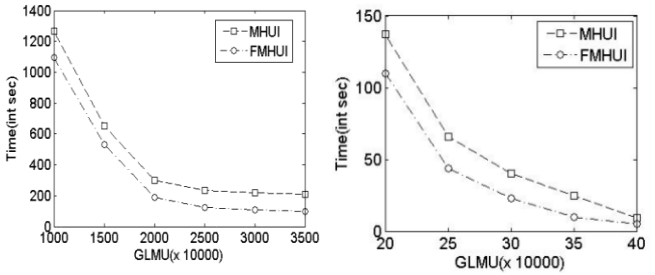
$$MU(x_i)=\max(\beta*\mathrm{EU}(x_i),GLMU) \quad (14)$$

其中：GLMU（global least minimum utility thresholds）是一个用户设定的全局最小效用阈值；β 是一个恒定的值来设定每个项的最小效用阈值，当 β 为 0 时，每个项的效用阈值都是 GLMU，就变成人们最常见的单一效用阈值了。

3.2　实验结果分析

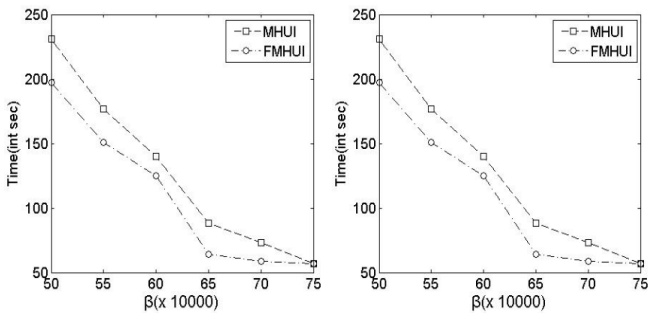
本文提出的算法主要验证加上支持度约束时挖掘频繁高效用项集算法的可行性以及算法效率的提高。因为在内存使用方面并没有明显的减少，所以这里就不做分析了。首先分析 FMHUI 与 MHUI 算法，其仿真结果如图 3 和 4 所示。

其中图 3 是 β 为常量，在 Accidents 和 Mushroom 中，β 分别为 500 万和 10 万。与 MHUI 算法相比，FMHUI 算法的时间效率平均提高 25.4%和 30.6%。图 4 是以 GLMU 为常量，在 Accidents 和 Mushroom 中，GLMU 分别为 1 200 万和 10 万，其时间效率分别提高 14%和 14.8%。这是因为 FMHUI 利用定义 8、11 减少了重复计算，提高了运行效率。



(a)Accidents　　(b)Mushroom

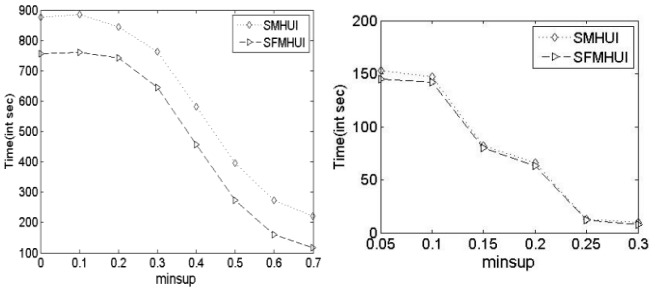
图 3　不同 GLMU 下算法的运行时间



(a)Accidents　　(b)Mushroom

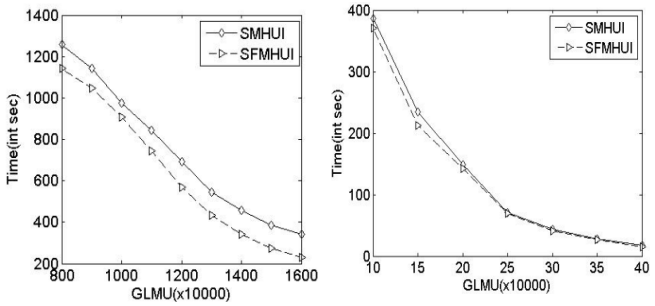
图 4　不同 β 下算法的运行时间

图 5 和 6 是 SFMHUI 算法和 SMHUI 算法的实验结果。因为这里主要是验证加上支持度约束时两种算法的实验结果，所以为了方便只考虑 β 和 GLMU 两个中的一个，这里只考虑了 GLMU。把 Accidents 和 Mushroom 的 β 分别设为 500 万和 5 万。图 5 是在 GLMU 分别为 1 400 万和 10 万时，SFMHUI 和 SMHUI 算法以最小支持度(minsup)为变量的情况下的运行时间。图 6 是在相同支持度时，改变 GLMU 值算法的运行时间。此时 Accidents 和 Mushroom 的支持度分别为 0.46 和 0.1。从这两个图可以看出，SFMHUI 比 SMHUI 算法更高效，提出的改进算法进一步得到验证。



(a)Accidents　　(b)Mushroom

图 5　不同 minsup 下算法的运行时间



(a)Accidents　　(b)Mushroom

图 6　不同 GLMU 下算法的运行时间

图 7 和 8 分析了在两个数据库中不同支持度下频繁高效用项集的数量。当 minsup=0 时，相当于没有支持度约束，然后随着支持度阈值约束 minsup 的增加，频繁高效用项集的数量在不断减少，验证了 SFMHUI 算法的可行性。有了支持度约束，挖掘的高效用项集对决策者来说更有参考价值。

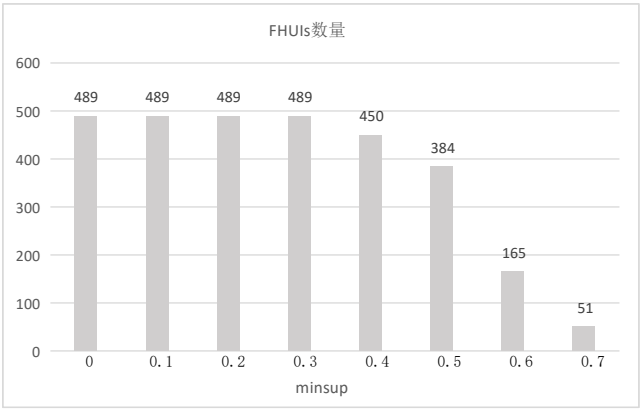


图 7　在 Accidents 中不同支持度下 FHUIs 数量

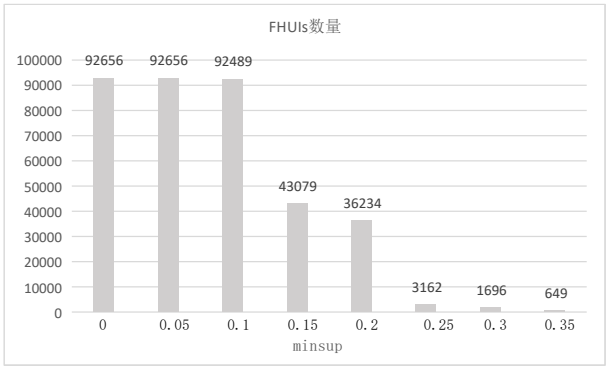


图 8 在 Mushroom 数据库中不同支持度下 FHUIs 数量

4 结束语

本文提出的 FMHUI 算法解决了多最小效用阈值挖掘算法 MHUI 存在的重复计算、重复比较问题；提出的 SFMHUI 算法在 FMHUI 算法的基础上增加支持度约束，进一步解决了挖掘的项集不是频繁项集的问题。两种算法都通过四个剪枝性质来提高挖掘效率，最后通过仿真实验进行了验证。但是本文在内存占用方面没有得到明显的改进，未来可以进一步结合并行计算减少内存消耗、提高算法运行效率；还可以考虑结合前缀效用树的节点列表 PUN-list（(prefix utility tree-node list）结构来提高挖掘效率，这些将是笔者下一步研究的方向和重点。

参考文献：

[1] Chan R, Yang Qiang, Shen Yidong. Mining high utility itemsets [C]// Proc of IEEE International Conference on Data Mining. Washington DC: IEEE Computer Society, 2003: 19-29.

[2] Duong Q H, Liao Bo, Fournier-Viger P, *et al.* An efficient algorithm for mining the top-k, high utility itemsets, using novel threshold raising and pruning strategies [J]. Knowledge-Based Systems, 2016, 104: 106-122.

[3] Lin Chunwei, Gan Wensheng, Fournier-Viger P, *et al.* High utility-itemset mining and privacy-preserving utility mining [J]. Perspectives in Science, 2016, 7: 74-80.

[4] Tseng V S, Wu Chengwei, Shie B E, *et al.* UP-Growth: an efficient algorithm for high utility itemset mining [C]// Proc of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Washington DC: IEEE Computer Society, 2010: 253-262.

[5] Zida S, Fournier-Viger P, Lin Chunwei, *et al.* EFIM: a fast and memory efficient algorithm for high-utility itemset mining [J]. Knowledge & Information Systems, 2017, 51 (2): 595-625.

[6] Thilagu M, Nadarajan R. Efficiently mining of effective Web traversal patterns with average utility [J]. Procedia Technology, 2012, 6 (4): 444-451.

[7] Lee D, Park S H, Moon S. Utility-based association rule mining: a marketing solution for cross-selling [J]. Expert Systems with Applications, 2013, 40 (7): 2715-2725.

[8] Lin Chunwei, Fournier-Viger P, Gan Wensheng. FHN: an efficient algorithm for mining high-utility itemsets with negative unit profits [J]. Knowledge-Based Systems, 2016, 111: 283-298.

[9] Ahmed C F, Tanbeer S K, Jeong B S, *et al.* HUC-Prune: an efficient candidate pruning technique to mine high utility patterns [J]. Applied Intelligence, 2011, 34 (2): 181-198.

[10] Tseng V S, Shie B E, Wu Chengwei, *et al.* Efficient algorithms for mining high utility itemsets from transactional databases [J]. IEEE Trans on Knowledge & Data Engineering, 2013, 25 (8): 1772-1786.

[11] Yun U, Ryang H, Ryu K H. High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates [J]. Expert Systems With Applications, 2014, 41 (8): 3861-3878.

[12] Liu Mengchi, Qu Junfeng. Mining high utility itemsets without candidate generation [C]// Proc of ACM International Conference on Information and Knowledge Management. New York: ACM Press, 2012: 55-64.

[13] Fournier-Viger P, Wu Chengwei, Zida S, *et al.* FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning [C]// Proc of International Symposium on Methodologies for Intelligent Systems: Foundations of Intelligent Systems. Switzerland: Springer International Publishing, 2014: 83-92.

[14] Krishnamoorthy S. Pruning strategies for mining high utility itemsets [J]. Expert Systems with Applications, 2015, 42 (5): 2371-2381.

[15] Zida S, Fournier-Viger P, Lin Chunwei, *et al.* EFIM: a highly efficient algorithm for high-utility itemset mining [C]// Proc of Mexican International Conference on Artificial Intelligence: Advances in Artificial Intelligence and Soft Computing. Switzerland: Springer International Publishing, 2015: 530-546.

[16] Lin Chunwei, Gan Wensheng, Fournier-Viger P, *et al.* Efficient mining of high-utility itemsets using multiple minimum utility thresholds [J]. Knowledge-Based Systems, 2016, 113: 100-115.

[17] Han Jiawei, Kamber M, Pei Jian. 数据挖掘概念与技术 [M]. 范明, 孟小峰, 译. 3 版. 北京: 机械工业出版社, 2007: 157-170. (Han Jiawei, Kamber M, Pei Jian. Data mining concepts and techniques [M]. Ming Fan, Xiaofeng Meng, Trans. 3rd Edi. Beijing: China Machine Press, 2007: 157-170.)

[18] Gan Wensheng, Lin Chunwei, Fournier-Viger P, *et al.* More efficient algorithms for mining high-utility itemsets with multiple minimum utility thresholds [C]// Proc of the 27th International Conference on Database and Expert Systems Applications. Switzerland: Springer International Publishing, 2016: 71-87.

[19] Krishnamoorthy S. Efficient mining of high utility itemsets with multiple minimum utility thresholds [J]. Engineering Applications of Artificial Intelligence, 2018, 69: 112-126.

[20] 茹蓓, 贺新征. 减少候选项集的数据流高效用项集挖掘算法 [J]. 计算机应用研究, 2017, 34 (11): 3379-3383. (Ru Bei, He Xinzheng. High utility itemsets mining algorithm of data stream with reducing candidate itemsets [J]. Application Research of Computers, 2017, 34 (11): 3379-3383)

[21] Sahoo J, Das A K, Goswami A. An efficient approach for mining

association rules from high utility itemsets [J]. Expert System with Applications, 2015, 42 (13): 5754-5778.

[22] Lin Chunwei, Gan Weisheng, Fournier-Viger P, *et al.* Efficient algorithms for mining high-utility itemsets in uncertain databases [J]. Knowledge-Based Systems, 2016, 96: 171-187.

[23] Fournier-Viger P, Gomariz A, Soltani A, *et al.* 2014a. SPMF: OpenSource data mining platform [EB/OL]. http://www. philippe-fournier-viger. com/spmf.